CS354: Machine Organization and Programming

Lecture 26
Monday the November 02nd 2015

Section 2
Instructor: Leo Arulraj
© 2015 Karen Smoler Miller
© Some examples, diagrams from the CSAPP text by Bryant and O’Hallaron

Class Announcements

1. Programming Assignment 3 is due by 9 AM day after tomorrow (11/04 – Wednesday). As usual, you can submit it up to 48 hours after the deadline with penalties.

Lecture Overview

1. Interrupts and Exception
2. Intro to Processes
1. User types 'K'

2. 'Q' appears in memory

1. 'Q' appears in memory

2. Character appears on display

The keyboard driver code:
```c
movl Keyboard.Data, eax

/* This is where the ASCII code will be after it's converted. */
```

The display driver code:
```c
movl eax, Display.Data
ret
```

What happens if user has not typed a char on the keyboard?

We want blocking input:
(No abortion until there is a char)

We need a status bit:
- 1 ready
- 0 not ready (lazy)

Place this bit into it's own memory mapped word so that the code can test for 0 or 1.
1. user types 'X'

2a. 'X' into Keyboard_Data
2b. 1 into Keyboard_Status

Now, driver code uses a spin wait loop (to implement blocking I/O)

```bash
```

```bash
disp.spin C:if Display.State,Display.State Jz disp.spin move Display.Data, to return
```

```bash
```

controller
P M
```

Disk
```
CPU disk controller USB controller memory
```
```
disk controller mouse keyboard
```
```
printer monitor graphics adapter
```
```
Byte transfers are OK, but, what about faster devices that like to transfer more than a byte?

The solution: DMA
Direct Memory Access

Issue for spin wait loop implementations:

One byte only in Data has the potential for an incorrect result.

For example, if the user types 2 characters on the keyboard before getchar() is called.

The needed fix introduces a kernel-maintained queue for each device.

Then, the kernel polls to check status bits and handle any ready devices.

Here is an analogy...

Teacher is OS
Each student is I/O device

Consider the inefficiency of OS polling.
An exception is an abrupt change in control flow.
Examples: div by 0, arithmetic overflow, page fault, I/O request completes, Ctrl-C
Classes of Exceptions

1. **Interrupts (Asynchronous):** Always return to next instruction.

2. **Traps & System Calls (Synchronous):** Always return to next instruction.

3. **Faults (Synchronous):** Might return to next instruction.

4. **Aborts (Synchronous):** Never returns

---

Interrupts

Examples of Interrupts:
- Timer interrupt
- Arrival of a packet from a network
- When a key is pressed on the keyboard
- When the mouse is moved

---

Traps

Traps are intentionally issued by executing an instruction.

Example: System calls

---

Faults

Faults result from error conditions that might be correctable.

Examples: Page fault, Divide error
Aborts

Aborts result from unrecoverable fatal errors.
Example: parity errors due to DRAM bit corruption

1. Fatal hardware error occurs
2. Control passes to handler
3. Abort handler runs
4. Handler returns to abort routine

Exception Table

Exception table

IA32 Exception Table

From CSAPP text book:

<table>
<thead>
<tr>
<th>Exception Number</th>
<th>Description</th>
<th>Exception Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Divide error</td>
<td>Fault</td>
</tr>
<tr>
<td>13</td>
<td>General protection fault</td>
<td>Fault</td>
</tr>
<tr>
<td>14</td>
<td>Page fault</td>
<td>Fault</td>
</tr>
<tr>
<td>18</td>
<td>Machine check</td>
<td>Abort</td>
</tr>
<tr>
<td>32-127</td>
<td>OS-defined</td>
<td>Interrupt or trap</td>
</tr>
<tr>
<td>128 [0x80]</td>
<td>System call</td>
<td>Trap</td>
</tr>
<tr>
<td>129-255</td>
<td>OS-defined</td>
<td>Interrupt or trap</td>
</tr>
</tbody>
</table>
Consider the x86 instruction:

cli clear IF

What happens if an application includes this cli instruction?

Irrelevant (to this discussion) x86 instruction:

sti set IF

OS relies on clock interrupts to allocate processing time. . .

As the clock interrupts, the kernel runs, and it decides which program runs next.
Clarified instruction:

cli  

\texttt{clear IF if CPL is high enough otherwise trap}

Does \texttt{sti} also need to be a privileged instruction?

---

Keep \[IF=1\] while \[CPL=00\].

So, applications can always be interrupted.

Hold most disable interrupts while

saving state \& at least until

first instruction within handler is

fetched.

**Reentrant Definitions:**

- \texttt{nonreentrant:} \[IF=0\] the entire time a handler runs.

- \texttt{reentrant:} interrupts may be

  enabled while handler runs

  (usually only for higher priority requests)

---

Non reentrant timeline

---

Reentrant timeline

dev 2 interrupts are higher priority